Attorney Docket No. NR-9 **Provisional Patent Application**

LEARNING STATE MACHINE FOR USE IN INTERNET PROTOCOL NETWORKS

Inventors:

Robert Daniel Maher, III
7401 Gurney Drive
Plano, Texas 75024

Aswinkumar Vishanji Rana
3213 Upshire Court
Plano, Texas 75075

Milton Andre Lie
5913 Spring Hill Drive
McKinney, Texas 75070

Travis Ernest Strother, Jr.
7501 Belmount Road
Rowlett, Texas 75089

Mark Warden Hervin
3605 Bent Ridge Drive
Plano, Texas 75074

James Robert Deerman
15 White Rock Trail
Lucas, Texas 75002

John Raymond Carman
90 Seattle Slew Lane
Fairview, Texas 75069

Larry Gene Maxwell
14439 Dennis Lane
Farmers Branch, Texas 75234

Craig J. Cox
Netrake Corporation
3000 Technology Drive, Suite 100
Plano, TX 70574
214.291.1050 phone
214.291.1010 fax

C:\Documents and Settings\carriew\My Documents\Patents\NR-9 Provisional\NR-9 Patent-Coversheet.doc

LEARNING STATE MACHINE FOR USE IN INTERNET PROTOCOL NETWORKS

TECHNICAL FIELD OF THE INVENTION

The present invention relates to broadband data networking equipment. Specifically, the present invention relates to a network processing system that is able to recognize events and to learn and maintain state for individual traffic flows on a broadband network.

BACKGROUND OF THE INVENTION

The power of internet protocol (IP) networks, such as the Internet, is their connectionless method of transporting data from source to destination. The nature of this connectionless transport is embodied in the "forward and forget" paradigm of the IP network's most powerful tool: the router. However, this greatest strength is also the IP network's greatest weakness. The "forward and forget" philosophy inherent in the network insures that there is no information about any data packet, or session, or flow is maintained by the network. Without information about packets or flows the network which could be used to allow the network to treat one packet or flow differently that the rest the network must treat every packet and flow the same resulting in the best efforts form of quality of service, which anyone who has ever used the Internet is familiar with.

To avoid the "forward and forget" paradigm, the network needs to be able to learn and maintain knowledge of the characteristics of the data packets and flows passing through it, in addition to the events that are contained within the contents of those data packets or flows. With this knowledge the network would have the information necessary to discriminate between packets and flows and give those with particular characteristics or events treatment different from the other packets in the network. For example, if the network was able to recognize a streaming video flow, the network could assign a higher quality of service to that flow to ensure that it passed through the network in the most efficient fashion. Similarly, if the network could recognize data packets from a user who had paid a premium for better service the network could ensure that those data packets received higher quality of service than packets from users who had not paid the premium. Further, if the network could

recognize events within flows, such as an email infected with a virus, the network could act on that information and discard the email or strip the infected attachment.

Accordingly, what is needed is a network processing system that can act as a learning state machine. The learning state machine being able to examine data packets and flows and learn characteristics about and events in those data packets and flows, save those characteristics and events as state, and to modify and direct the data packets and flows based, at least in part, on the state information.

SUMMARY OF THE INVENTION

The present invention provides for a network processing system that acts as a learning state machine to identify and learn characteristics about and events contained in the data packets and flows being processed. The network processing system includes a network interface, or line interface, which receives data in the form of packets from the network and transmits processed data, or packets, back onto the network. The network interface communicates with a learning state machine in the network processing system that is operable to assign an identifier to the packet. The identifier associates the packet with the particular session, or flow, of which the packet is a part. This identifier allows the learning state machine maintain a state database of characteristics and events related to that flow even across packet boundaries. Using any previously determined state and the packet being processed, the learning state machine compares the packet to a database of known signatures, the database of known signatures included programmable characteristics and events for which the network processing is programmed to respond. This comparison results in the processing engine determining a treatment for the packet. The treatment could include modification, and routing and/or the direction of the packets back onto the network. In one embodiment of the network processing system two unidirectional processing engines are used in communication with each other and a management module to produce a bi-directional network processing system.

The learning state machine or machines in the network processing system further include a traffic flow scanning processor, or traffic flow processor, which is operable to associate each packet with the identifier identifying its flow. The traffic flow processor compares each packet to the database of programmed signatures containing the known signatures, and determines a treatment for the packet. The traffic flow engine also maintains the state database. A quality of service processor

communicates with the traffic flow processor and uses the treatment to modify and direct the packet back onto the network.

A management interface is also described that resides on a server separate from the network processing system. The management interface includes the programming interface, either a command line interface or a graphical user interface, used to set the network policies and other signature and configuration information. The programming information is used in conjunction with other databases and libraries to build an image file that can be remotely loaded into the appropriate network processing system under the management interfaces control. The image file contains all the programmable information, including configuration and signature images necessary for the network processing system to operate as a policy gateway.

The foregoing has outlined, rather broadly, preferred and alternative features of the present invention so that those skilled in the art may better understand the detailed description of the invention that follows. Additional features of the invention will be described hereinafter that form the subject of the claims of the invention. Those skilled in the art will appreciate that they can readily use the disclosed conception and specific embodiment as a basis for designing or modifying other structures for carrying out the same purposes of the present invention. Those skilled in the art will also realize that such equivalent constructions do not depart from the spirit and scope of the invention in its broadest form.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, in which:

Figure 1 is a network topology diagram illustrating example network structures in which the present invention can operate;

Figure 2 is a diagram illustrating flow, packet and block concepts used in the present invention;

Figure 3 is a block diagram of a network processing system according to the present invention;

Figure 4 is a block diagram of the processing engines shown in Figure 3;

Figure 5 is a block diagram of the content processor from Figure 4;

Figure 6 is a diagram of the image builder used to create the image and configuration files used in the network processing system of the present invention; and

Figure 7 is a diagram showing the mechanism by which the image files are loaded into and statistical and event information are retrieved from the network processing system of the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

Referring now to Figure 1, a network topology is shown which is an example of network infrastructures that exist within a broader public IP network such as the internet. Figure 1 is in no way meant to be a precise network architecture, but only to serve as a rough illustration of a variety of network structures which can exist on a broadband IP network. Figure 1 shows a core IP network 10 which can be the IP network of a company such as MCI or UUNET, and an access network 12, which connects users through equipment such as DSLAMs 14 or enterprise routers 16 to the core IP network 10. An endless variety of network structures can be connected to core IP network 10 and access network 12 in order to access other networks connected to the public IP network, and these are represented here as clouds 18.

Access network 12, an example of which would be an Internet Service Providers (ISPs) or Local Exchange Carriers (LECs), is used to provide both data and voice access over the public IP network. Access network 12 can provide services for enterprises through enterprise routers 16, for example company networks such as the company network for Lucent Technologies or Merrill Lynch, or for individual homes, home offices, or small businesses through dial-up or high speed connections such as digital subscriber lines (DSL) which connect through aggregation devices such as DSLAM 14.

Access network 12 includes a switched backbone 20, shown here as an asynchronous transfer mode (ATM) network, which is formed by switches and routers, to route data over its network. Domain name servers and other networking equipment, which are not shown, are also included in access network 12. Access network 12 provides connections between its own subscribers and between its subscribers and core IP network 10 and other networks 16 so that its subscribers can reach the customers of other access networks.

It can easily be seen that points exist at the edges of the network structures and between network structures where data is passed across network boundaries. One major problem in the network structures shown in Figure 1 is the lack of any type of intelligence at these network boundary points which would allow the network to provide services such as quality of service, policy enforcement, security and statistical metering. The intelligence to provide these services would require that the network understand the type of data passing through these network boundary points and not just the destination and/or source information which is currently all that is understood.

Understanding the type of data, or its contents, including the contents of the associated payloads as well as header information, and further understanding and maintaining a state awareness across each individual traffic flow would allow the network to enforce network policies in real time, thereby allowing the network to provide real cross network QoS using standards such as MPLS and DiffServ, to configure itself in real time to bandwidth requirements on the network for applications such as VoIP or video where quality of service is a fundamental requirement, or to provide other network services which require intelligence at the session, or flow, level and not just packet forwarding. An intelligent network would also be able to identify and filter out security problems such as email worms, viruses, denial of service (DoS) attacks, and illegal hacking in a manner that would be transparent to end users. Further, the intelligent network would provide for metering capabilities by hosting companies and service providers, allowing these companies to regulate the amount of bandwidth allotted to individual customers as well as to charge precisely for bandwidth and additional features such as security.

An example of the employment of such a device is shown in Figure 1 by network processing system 22, which resides at the cross network boundaries as well as at the edge of the access network 12 behind the DSLAMs 14 or enterprise routers 16. A device at these locations would, if it were able to identify and track flows, and to maintain state for those flows, be able to provide real quality of service and policy management to networks to which it was connected.

In accordance with the requirements set forth above, the present invention provides for a network processing system that is able to scan, classify, and modify network traffic including payload information at speeds of OC-3, OC-12, OC-48 and greater thereby providing an effective policy management platform.

In order to help understand the operation of the network processing system described herein, Figure 2 is provided to illustrate concepts relating to network traffic that will be used extensively herein. Figure 2 shows three individual flows, Flow (NID-a), Flow (NID_b), and Flow (NID_c), which can be simultaneously present on the network. Each flow represents an individual session that exists on the network. These sessions can be real-time streaming video sessions, voice over IP (VoIP) call, web-browsing, file transfers, or any other network traffic. Each flow is made up of individual data packets, packets x and x+1 for Flow (NID_a), packets y and y+1 for Flow (NID_b) and packets z and z+1 for Flow (NID_c). While two packets are

shown, each flow is made up of an arbitrary number of packets and each packet is of an arbitrary size. Each packet can further be broken down into fixed length blocks shown for each packet as Blk_i, Blk_i+1, and Blk_i+2. While packets and flows appear as network traffic, the fixed length blocks shown in Figure 2 are created by the network processing system of the present invention and will be described with greater detail below.

Referring now to Figure 3, one embodiment of a network processing system according to the present invention is shown. Network processing system 40 is a bi-directional system that can process information from either right line interfaces 42 which is then transmitted back onto the network through left line interfaces 44, or from left line interfaces 44 which is then transmitted back onto the network through right lines interfaces 42. Both left and right line interfaces 44 and 42 respectively, can consist of any plurality of ports, and can accept any number of network speeds and protocols, including such high speeds as OC-3, OC-12, OC-48, and protocols including 10/100 Ethernet, gigabit Ethernet, and SONET.

The line interface cards take the incoming data in the form of packets and place the data on a data bus 54 which is preferably an industry standard data bus such as a POS-PHY Level 3, or an ATM UTOPIA Level 3 type data bus. Data received on left line interfaces 44 is sent to learning state machine, or processing engine 44 while data received on right line interfaces 42 is sent to learning state machine, or processing engine 46. While network processing system 40 is bi-directional, individual processing engines 44 and 46 within network processing system 40 are unidirectional requiring two to process bi-directional information. Each processing engine 44 and 46, the operation of which will be described in greater detail with reference to Figure 4, is operable to scan the contents of each data packet, associate the data packet with a particular flow, determine the treatment for each data packet based on its contents and any state for the associated flow, and queue and modify the data packet to conform to the determined treatment. The state for flows, is the information related to that flow that has been identified by network processing system 40 from packets associated with the flow that have already been processed.

An internal bus 52, which is preferably a PCI bus, is used to allow processing engines 44 and 46 to communicate with each other, and to allow management module 48 and optional auxiliary processor module 50 to communicate with both processing engines 44 and 46. Intercommunication between processing engines 44 and 46 allow

the processing engines to exchange information learned from a flow that can be applied to the treatment in the return flow. For example, treatment for a high-priority customer needs to be applied to both outgoing and incoming information. Since each processing engine is unidirectional, to affect both directions of traffic, information must be shared between processing engines.

Management module 48 is used to control the operation of each of the processing engines 44 and 46 and to communicate with external devices which are used to load network processing system 40 with policy, QoS, and treatment instructions that network processing system 40 applies to the network traffic it processes.

Referring now to Figure 4, one embodiment of a content processing engine used in the network processing system according to the present invention is shown. Each of the processing engines 44 and 46 are identical as discussed and the operation of each will be discussed generally and any description of the operation of the processing engines will apply equally to both processing engines 44 and 46. Line interface cards 42 and 44, shown in Figure 3, take the data from the physical ports, frames the data, and then formats the data for placement on fast-path data bus 126 which, as described, is preferably an industry standard data bus such as a POS-PHY Level 3, or an ATM UTOPIA Level 3 type data bus.

Fast-path data bus 126 feeds the data to traffic flow scanning processor 140, which includes header preprocessor 104 and content processor 110. The data is first sent to header preprocessor 104, which is operable to perform several operations using information contained in the data packet headers. Header preprocessor 104 stores the received data packets in a packet storage memory associated with header preprocessor 104, and scans the header information. The header information is scanned to identify the type, or protocol, of the data packet, which is used to determine routing information and to decode the IP header starting byte. As will be discussed below, the processing engine, in order to function properly, needs to reorder out of order data packets and reassemble data packet fragments. Header preprocessor 104 is operable to perform the assembly of asynchronous transfer mode (ATM) cells into complete data packets (PDUs), which could include the stripping of ATM header information.

After data packets have been processed by header preprocessor 104 the data packets, any conclusion formed by the header preprocessor, such as QoS information,

are sent on fast-data path 126 to the other half of traffic flow scanning engine 140, content processor 110. The received packets are stored in packet storage memory 112 while they are processed by content processor 110. Content processor 110 is operable to scan the contents of data packets received from header preprocessor 104, including the entire payload contents of the data packets. The header is scanned as well, one goal of which is to create a session id using predetermined attributes of the data packet.

In the preferred embodiment, a session id is created using session information consisting of the source address, destination address, source port, destination port and protocol, although one skilled in the art would understand that a session id could be created using any subset of fields listed or any additional fields in the data packet without departing from the scope of the present invention. When a data packet is received that has new session information the header preprocessor creates a unique session id to identify that particular traffic flow. Each successive data packet with the same session information is assigned the same session id to identify each packet within that flow. Session ids are retired when the particular traffic flow is ended through an explicit action, or when the traffic flow times out, meaning that a data packet for that traffic flow has not been received within a predetermined amount of time. While the session id is discussed herein as being created by the header preprocessor 104 the session id can be created anywhere in traffic flow scanning engine 140 including in content processor 110.

The contents of any or all data packets are compared to a database of known signatures and if the contents of a data packet, or packets, match a known signature, an action associated with that signature and/or session id can be taken by the processing engine. Additionally, content processor 110 is operable to maintain state awareness throughout each individual traffic flow. In other words, content processor 110 maintains a database for each session which stores state information related to not only the current data packets from a traffic flow, but state information related to the entirety of the traffic flow. This allows network processing system 100 to act on not only based on the content of the data packets being scanned but also based on the contents of the entire traffic flow. The specific operation of content processor 110 will be described with reference to Figure 5.

Once the contents of the packets have been scanned and a conclusion reached by traffic flow scanning engine 140, the packets and the associated conclusions of

either or both the header preprocessor 104 and the content processor 110 are sent to quality of service (QoS) processor 116. QoS processor 116 again stores the packets in its own packet storage memory for forwarding. QoS processor 116 is operable to perform the traffic flow management for the stream of data packets processed by network processing system 40. QoS processor contains engines for traffic management, traffic shaping and packet modification.

QoS processor 116 takes the conclusion of either or both of header preprocessor 104 and content processor 110 and assigns the data packet to one of its internal quality of service queues based on the conclusion. The quality of service queues can be assigned priority relative to one another or can be assigned a maximum or minimum percentage of the traffic flow through the device. This allows QoS processor 116 to assign the necessary bandwidth to traffic flows such as VoIP, video and other flows with high quality and reliability requirements while assigning remaining bandwidth to traffic flows with low quality requirements such as email and general web surfing to low priority queues. Information in queues that do not have the available bandwidth to transmit all the data currently residing in the queue according to the QoS engine is selectively discarded thereby removing that data from the traffic flow.

The quality of service queues also allow network processing system 40 to manage network attacks such as denial of service (DoS) attacks. Network processing system 40 can act to qualify traffic flows by scanning the contents of the packets and verifying that the contents contain valid network traffic between known sources and destinations. Traffic flows that have not been verified because they are from unknown sources or because they are new unclassified flows can be assigned to a low quality of service queue until the sources are verified or the traffic flow is classified as valid traffic. Since most DoS attacks send either new session information, data from spoofed sources, or meaningless data, network processing system 40 would assign those traffic flows to low quality traffic queues. This ensures that the DoS traffic would receive no more than a small percentage (i.e. 5%) of the available bandwidth thereby preventing the attacker from flooding downstream network equipment.

The QoS queues in QoS processor 116 (there are 65k queues in the present embodiment of the QoS processor although any number of queues could be used) feed into schedulers (1024 in the present embodiment), which feed into logic ports (256 in the present embodiment), which send the data to flow control port managers 138 (32

in the present embodiment) which can correspond to physical egress ports for the network device. The traffic management engine and the traffic shaping engine determine the operation of the schedulers and logic ports in order to maintain traffic flow in accordance with the programmed parameters.

QoS processor 116 also includes a packet modification engine, which is operable to modify, add, or delete bits in any of the fields of a data packet. This allows QoS processor 116 to change DiffServ bits or to place the appropriate MPLS shims on the data packets for the required treatment. The packet modification engine 130 can also be used to change information within the payload itself if necessary. Data packets are then sent along fast-data path 126 to output to the associate line interfaces where it is converted back into an analog signal and placed on the network.

As with all network equipment, a certain amount of network traffic will not be able to be processed along fast-data path 126. This traffic will need to be processed by on-board microprocessor 124. The fast-path traffic flow scanning engine 140 and QoS processor 116 send packets requiring additional processing to flow management processor 122, which forwards them to microprocessor 124 for processing. The microprocessor 124 then communicates back to traffic flow scanning engine 140 and QoS processor 116 through flow management processor 122. Flow management processor 122 is also operable to collect data and statistics on the nature of the traffic flow through the processing engine 100. Bridges 146 are used between elements to act as buffers on PCI buses 148 in order to prevent the loss of data that could occur during a flood of the PCI bus.

As can be seen from the description of Figure 4, processing engines 44 and 46 allow the entire contents of any or all data packets received to be scanned against a database of known signatures. The scanned contents can be any variable or arbitrary length and can even cross packet boundaries. The abilities of processing engines 44 and 46 allow the construction of a network device that is intelligent which gives the network device the ability to operate on data packets based on the content of that data packet.

Referring now to Figure 5, the content processor 110 of Figure 4 is described in greater detail. As described above, content processor 110 is operable to scan the contents of data packets forwarded from header preprocessor 104 from Figure 4. Content processor 110 includes three separate engines, queue engine 302, context engine 304, and content scanning engine 306.

Since content processor 110 scans the contents of the payload, and is able to scan across packet boundaries, content processor 110 must be able to reassemble fragmented packets and reorder out of order packets on a per session basis. Reordering and reassembling is the function of queue engine 302. Queue engine 302 receives data off the fast-path data bus 126 using fast-path interface 310. Packets are then sent to packet reorder and reassembly engine 312, which uses packet memory controller 316 to store the packets into packet memory 112. Reordering and reassembly engine 312 also uses link list controller 314 and link list memory 318 to develop detailed link lists that are used to order the data packets for processing. The data packets are broken into 256 byte blocks for storage within the queue engine 302. Session CAM 320 can store the session id generated by queue engine 302 of content processor 110. Reordering and reassembly engine 312 uses the session id to link data packets belonging to the same data flow.

In order to obtain the high throughput speeds required, content processor 110 must be able to process packets from multiple sessions simultaneously. Content processor 110 processes blocks of data from multiple data packets each belonging to a unique traffic flow having an associated session id. In the preferred embodiment of the present invention, context engine 304 of content processor 110 processes 64 byte blocks of 64 different data packets from unique traffic flows simultaneously. Each of the 64 byte blocks of the 64 different data flows represents a single context for the content processor. The scheduling and management of all the simultaneous contexts for content processor 110 is handled by context engine 304.

Context engine 304 works with queue engine 302 to select a new context when a context has finished processing and has been transmitted out of content processor 110. Next free context/next free block engine 330 communicates with link list controller 314 to identify the next block of a data packet to process. Since content processor 110 must scan data packets in order, only one data packet or traffic flow with a particular session id can be active at one time. Active control list 332 keeps a list of session ids with active contexts and checks new contexts against the active list to insure that the new context is from an inactive session id. When a new context has been identified, packet loader 340 uses the link list information retrieved by the next free context/next free block engine 330 to retrieve the required block of data from packet memory 112 using packet memory controller 316. The new data block is then

loaded into a free buffer from context buffers 342 where it waits to be retrieved by content scanning engine interface 344.

Content scanning engine interface 344 is the interface between context engine 304 and content scanning engine 306. When content scanning engine 306 has room for a new context to be scanned, content scanning engine interface 344 sends a new context to string preprocessor 360 in content scanning engine 306. String preprocessor 360 is operable to simplify the context by performing operations such as compressing white space (i.e. spaces, tabs, returns) into a single space to simplify scanning. Once string preprocessor 360 has finished, the context is loaded into one of the buffers in context buffers 362 until it is retrieved by string compare 368. String compare 368 controls the input and output to signature memory 366. While four signature memories 366, each of which is potentially capable of handling multiple contexts, are shown any number could be used to increase or decrease the throughput through content scanning engine 110. In the present embodiment, each of the signature memories 366 is capable of processing four contexts at one time.

One of the signature memories 366 is assigned the context by scheduler 364 and then compares the significant bits of the context to the database of known strings that reside in signature memory 366. The signature memory 366 determines whether there is a potential match between the context and one of the known signatures using significant bits, which are those bits that are unique to a particular signature. If there is a potential match, the context and the potentially matched string are sent to leaf string compare 368 which uses leaf string memories 370 to perform a bit to bit comparison of the context and the potentially matched string. Although four string memories 366 and two leaf string memories 370 are shown, any number of string memories 366 and leaf string memories 370 can be used in order to optimize the throughput of content processor 110.

The conclusion of the content scanning are then sent back to the payload scanning interface 344 along with possibly a request for new data to be scanned. The conclusion of the content scanning can be any of a number of possible conclusions. The scanning may not have reached a conclusion yet and may need additional data from a new data packet to continue scanning in which case the state of the traffic flow, which can be referred to as an intermediate state, and any incomplete scans are stored in session memory 354 along with other appropriate information such as sequence numbers, counters, etc. The conclusion reached by signature memory 366

may also be that scanning is complete and there is or isn't a match, in which case the data packet and the conclusion are sent to transmit engine 352 for passing to QoS processor 116 from Figure 4. The scanning could also determine that the data packet needs to be forwarded to microprocessor 124 from Figure 4 for further processing, so that the data packet is sent to host interface 350 and placed on host interface bus 372. In addition to handling odd packets, host interface 350 allows microprocessor 124 to control any aspect of the operation of content processor 110 by letting microprocessor 124 write to any buffer or register in context engine 304.

State information is stored in session memory 354 and is updated as necessary after data associated with the particular traffic flow is scanned. The state could be an intermediate state, representing that the matching is incomplete and additional data is needed to continue the scanning. Also, the state could be a partial state indicating that one or more events have occurred from a plurality of events required to generate a particular conclusion. The state may be a final state indicating that a final conclusion has been reached for the associated traffic flow and no further scanning is necessary. Or, the state may represent any other condition required or programmed into the content processor 110. The state information for each traffic flow, in whatever form, represents the intelligence of network processing system 40 from Figure 3, and allows the network processing system to act not only on the information scanned, but also on all the information that has been previously scanned for each traffic flow.

The operation of transmit engine 352, host interface 350, session memory controller 348, which controls the use of session memory 354, and of general-purpose arithmetic logic unit (GP ALU) 346, which is used to increment or decrement counter, move pointers, etc., is controlled by script engine 334. Script engine 334 operates to execute programmable scripts stored in script memory 336 using registers 338 as necessary. Script engine 334 uses control bus 374 to send instruction to any of the elements in context engine 304. Script engine 334 or other engines within content processor 110 have the ability to modify the contents of the data packets scanned. For example, viruses can be detected in emails scanned by content processor 110, in which case the content processor can act to alter the bits of infected attachment essentially rendering the email harmless.

The abilities of content processor 110 are unique in a number of respects. Content processor 110 has the ability to scan the contents of any data packet or packets for any information that can be represented as a signature or series of

signatures. The signatures can be of any arbitrary length, can begin and end anywhere within the packets and can cross packet boundaries. Further, content processor 110 is able to maintain state awareness throughout all of the individual traffic flows by storing state information for each traffic flow representing any or all signatures matched during the course of that traffic flow. Existing network processors operate by looking for fixed length information at a precise point within each data packet and cannot look across packet boundaries. By only being able to look at fixed length information at precise points in a packet, existing network processors are limited to acting on information contained at an identifiable location within some level of the packet headers and cannot look into the payload of a data packet much less make decisions on state information for the entire traffic flow or even on the contents of the data packet including the payload.

Referring now to Figure 6, a diagram of the software that creates the processor configurations and most importantly the memory images that form the database of signatures in the content processor 110 to which each packet and flow is compared. The software used to build the memory images and configurations is run on a server separate from the network processing system described in Figure 3. Once created on the separate server, the memory images and configurations are transmitted and downloaded into the network processing system as will be described with reference to Figure 7.

The network processing system of Figure 3 is programmable by a user to set the network policies, which it will enforce. The programming is done using policy image builder 500, which is loaded on a separate server, as described. Policy image builder 500 includes a graphical user interface (GUI) 502, and a command line interface (CLI) 504. The functionality of the GUI 502 and CLI 504 are identical and are provided to allow the programmer to choose a preferred interface. A policy gateway configuration database 510 holds information relating to the configuration of each policy gateway, including such information as memory sizes, port numbers, type of line interfaces, etc., to which the programmer has access, and interacts with the CLI interpreter 508 and GUI program 506 to send the new user program to databases holding existing processing engine configuration files 514 and existing policy descriptions 512. The new user program and the existing configurations and descriptions are then combined with object libraries 518 by Policy Object Language (POL) Constructor 516. POL Constructor 516 takes the program and configuration

information and produces several maps and configuration files for the individual components of the network processing system.

First, a map of the memory locations inside the network processing engine is produced and stored in memory and counter map 520. Since the network processing system is fully programmable, individual memory locations, counters and registers are assigned functionality by the program. Without a map of the assignments, the data subsequently read from the network processing system would be unintelligible. The memory and counter map produced allows any data produced by the network processing system to be interpreted later.

Additionally, the POL Constructor 516 produces the configuration files for each of the network processing system components. A QoS configuration file 528 is produced that is sent to a QoS compiler 530 and used to produce a QoS configuration image 546. A Header Preprocessor (HPP) program 526 is produced and sent to a HPP compiler 532, which produces an HPP binary file 544. Similarly, a Context Engine script file 524 is produced by POL Constructor 516, which is compiled by context engine script compiler 534 to produce context engine binary file 542. Finally, a signature map file 522 is created that includes the network policy description, and sent to signature algorithm generator 536 which compresses the signature map into an efficient signature memory map 540 in order to more efficiently use the memory in the network processing system. The program also allows for partial updates of the signature memory by using a partial signature memory map 538, which can be used to change only a small part of the signature memory if a full remap of the signature memory is unnecessary.

These four binary files, the QoS configuration file 528, the HPP binary file 544, the context engine binary file 542 and the signature memory map 540 (or partial signature memory map 538, as appropriate) are then combined, along with the processing engine configure source file 552, the policy description source file 550 and the counter and memory map source file 548. The combination is done by the processing engine image builder 554, which produces a policy gateway image load file 556. The policy gateway image load file 556 is the file sent from the separate server to the actual network processing systems to provide the networking processing system with the information and programs necessary to run. The source files are included in the policy gateway image load file 556 to allow the four binary files to be reconstructed and understood from the policy gateway image load file alone, without

having to retrace source files in other locations, should anything happen to any part of the network or system.

To understand exactly what is contained in the policy gateway image file 556, the individual components are illustrated as processing engine data 558, control processor data 560, and management processor data 562. Processing engine data 558 contains the left and right signature memory maps for both the left and right processing engines 44 and 46 from Figure 3, which are loaded into the signature memory of content processors 110 shown in Figure 4. Processing engine data 558 also contains the left and right configuration files for QoS processors 116 for left and right processing engines 44 and 46, respectively, as shown in Figure 4. Finally processing engine data 558 contains the left and right header preprocessor image files for header preprocessors 104 for left and right processing engine 44 and 46 respectively.

Control processor data 560 contains left and right counter memory maps which are loaded into microprocessor 124 on each of left and right processing engines, respectively. Finally, management processor data 562 contains the left and right configuration source and the left and right policy source, as described above with reference to processing engine configuration source 552 and policy source 550. These files are stored on management module 48 shown in Figure 3.

Referring now to Figure 7, a diagram showing the mechanics of communication with the network processing systems is described. The programs implementing the diagram shown in Figure 7 also reside on the separate server that includes policy image builder 500 described in Figure 6. As described above, CLI 504 and GUI 506 are used with configuration files 510 by policy image builder 500 to produce both policy gateway image file 556 and memory and counter map 520. Policy gateway image file 556 is taken by image repository manager 570 and loaded into image repository database 572. Image repository database 572 holds all the policy gateway image files for all of the network processing systems being controlled. Network processing system (NPS) interface program 580 is responsible for the direct communication with each of the network processing systems NPS #001, NPS #002, and NPS #00n being managed. As indicated by NPS#00n, any number of network processing systems can be managed from one separate server. Image repository program 574 takes the proper image file from image repository database 572 and sends it to NPS interface program 580. NPS interface program 580 acts to

authenticate each network programming system using authentication program 584 and then sends the policy gateway image file to the appropriate network processing system.

In addition to pushing image files to the network processing systems, NPS interface program 580 acts to pull statistical and event data out of each network processing system by periodically sending each network processing system requests to upload its statistical and event information. When this information is received by NPS interface program it is sent to statistical database manage 586, which stores it in statistics database 588. Statistics database manager 590 uses information out of memory and counter map 520 to place the information necessary to decipher statistics database 588 into statistics configuration database 592. Statistics database 588 and statistics configuration database 592 can then be used to feed information into billing systems to bill for services, and into network management systems to analyze network operations and efficiency.

While the header preprocessor, the QoS processors, and the flow management processor described with reference to Figures 3 and 4 can be any suitable processor capable of executing the described functions, in the preferred embodiment the header preprocessor is the Fast Pattern Processor (FPP), the QoS processor is the Routing Switch Processor (RSP), and the flow management processor is the ASI processor, all manufactured by the Agere Division of Lucent Technologies, Austin Texas. The microprocessor described with reference to Figures 3 and the management module of Figure 4 could be any suitable microprocessor including the PowerPC line of microprocessors from Motorola, Inc., or the X86 or Pentium line of microprocessors available from Intel Corporation. Although particular references have been made to specific protocols, implementations and materials, those skilled in the art should understand that the network processing system, both the "bump-in-the-line" and the routing apparatus can function independent of protocol, and in a variety of different implementations without departing from the scope of the invention.

Although the present invention has been described in detail, those skilled in the art should understand that they can make various changes, substitutions and alterations herein without departing from the spirit and scope of the invention in its broadest form.